MIT LIBRARIES DUPL
3 9080 00698826 2

CHUSETTS
LIBRARIES
TITUTE of TECHNOLOGY

DEWE

MASS. INST. TECH.
OCT 1 1990

WORKING PAPER
ALFRED P. SLOAN SCHOOL OF MANAGEMENT

AN IMPROVED STRUCTURAL TECHNIQUE FOR AUTOMATED RECOGNITION OF HANDWRITTEN INFORMATION

Patrick S-P Wang
Amar Gupta

August 1990 WP# 3162-90

MASSACHUSETTS
INSTITUTE OF TECHNOLOGY
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139

AN IMPROVED STRUCTURAL TECHNIQUE FOR AUTOMATED RECOGNITION OF HANDWRITTEN INFORMATION

Patrick S-P Wang
Amar Gupta

August 1990 WP# 3162-90

AN IMPROVED STRUCTURAL TECHNIQUE FOR AUTOMATED RECOGNITION OF HANDWRITTEN INFORMATION

Patrick S-P Wang
Amar Gupta

IFSRC No. 123-90

Comments and suggestions relating to the contents of this paper should be directed to the principal investigator:

Dr. Amar Gupta
Principal Research Associate
Sloan School of Management
MIT Room E40-265
Cambridge, MA 02139
Telephone: 617-253-8906

Abstract

This paper examines several line-drawing pattern recognition methods for handwritten character recognition such as picture descriptive language(PDL), Berthod and Maroy's method(BM), extended Freeman's chain code(EFC), error transformation(ET)method, tree grammar(TG), and array grammar(AG). Then a new character recognition scheme that uses improved extended octal codes as the primitives is introduced. This scheme offers advantages of handing flexible size, orientation, and variations, need for fewer learning samples needed and lower degree of ambiguity. Finally the simulation of off-line character recognition by their real time on-line counterpart is investigated.

Keywords : *pattern recognition, character recognition, syntactic approach, structural approach, grammars, on-line vs off-line, learning, degrees of recognizability, learnability and ambiguity, artificial intelligence*

1. Introduction

During the past quarter century there has been a growing interest for computer scientists, engineers and researchers in the problems related to the replication of human function and machine simulation of human reading. Intensive research has been done in this area and a large number of technical papers and reports on line-drawing pattern recognition (including character recognition) can be found in the literature[1,2,3,8,10,12-16,23-25,27-32]. This subject has attracted such an immense interest and effort not only because it is very interesting and challenging, but also because it provides a strategy for automated postal code reading, check verification, office automation and a large variety of banking, business, industrial, engineering and scientific applications[8,25,27].

Recently, the state of the art in line-drawing pattern recognition has advanced from the use of primitive schemes for the recognition of machine-printed alphanumerals to the application of sophisticated techniques for the recognition of a wide variety of handwrit-

ten characters and symbols [1,27,32,34].

It is interesting to see that even human beings, who possess the best trained optical reading devices (eyes), make about 5-percent mistakes when reading in the absence of context[24]. These errors are mainly caused by infinite variations of shapes resulting from the writing habit, styles, education, region of origin, social environment, mood, health and other conditions of the writer. For character recognition by computers, the error rate could be even higher because of the reasons described above as well as other factors such as the writing instrument, writing surface, scanning mechanisms, learning techniques and machine recognition algorithms[3,8].

This paper begins with an analysis and comparison of several line-drawing pattern recognition methods. These approaches can be basically classified into two categories, namely (1) syntactic approach (such as PDL, tree grammars and array grammars) and (2) structural approach (such as BM method and extended Freeman's chain code method). Their advantages and disadvantages are discussed. The reason we concentrate on syntactic and structural methods rather than conventional statistical methods, is because they are hierarchical in nature, i.e. they divide a large, complicated pattern into smaller subpatterns according to its structure, which in turn can be further divided into smaller subpaterns and so on, till primitives are found which can no longer be divided. Using such structural and syntactic approaches, one can directly take advantages of the powerful data structures using grammatical rules, trees and directed labeled graphs, which are widely used in dealing with linguistic problems [6,7,11, 18,20,21]. Such approaches are similar to the powerful artificial intelligence problem solving str tegies in which a large or complicated problem is reduced to smaller subproblems [17,19,26,35,36]. Further, it has been shown that syntactic and structural approaches can overcome some disadvantages found in classic decision theoretic (statistical) approach, which has difficulty in distinguishing between two very similar patterns (characters) [6,18]. Among the various structural and syntactic approaches, we believe that those with a higher degree of recognizability are normally more natural in that they are more accurate and closer to the intuition of human beings, who possess the best pattern recognition skill for dealing with handwritten symbols.

A new character recognition scheme using improved extended octal code as primitives is discussed. This scheme provides certain advantages such as flexible size, orientation, variations, need for fewer learning samples and lower degree of ambiguity and higher degree of recognizability. Finally, an off-line character recognition technique that simulates on-line real time techniques is studied.

2. Notations, Terminologies, Definitions, and Fundamentals

Frequently we hear people say, "*Your handwriting is terrible. It is hard to recognize.*" But what constitutes a well recognizable or a hardly recognizable handwriting at the level of character, word or even a sentence? Looking at Table 2.1, one can see that category 1 is recognizable, category 2 is recognizable with some effort, category 3 is recognizable with great difficulty (and perhaps with a high risk of mis-recognition) and category 4 is beyond recognition. While this is intuitively true, it is necessary to define a more concrete criterion (or criteria) of "recognizable" characters for machine simulation of human reading. Figure 2.1 shows a block diagram of a typical character recognition scheme.

Definition 2.1 The degree of ambiguity of a character B can be defined in terms of the problem domain D and the encoding scheme E as follows :

DegAmb(B,D,E)=max. no. of interpretations of B in D under E.

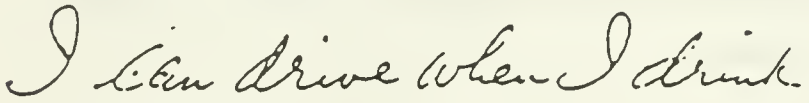
Definition 2.2 The degree of recognizability of a character B, in the domain of D, under the encoding scheme E can be defined as follows:

$$DegRec(B,D,E) = \begin{cases} 1/DegAmb(B,D,E) & \text{if B is in the dictionary} \\ 0 & \text{otherwise} \end{cases}$$

Conventionally one would conjecture that the easier a character is to learn the easier it is to recognize it and vice versa. However, this is not always true. We do find examples

Table 2.1
Several levels of recognizability

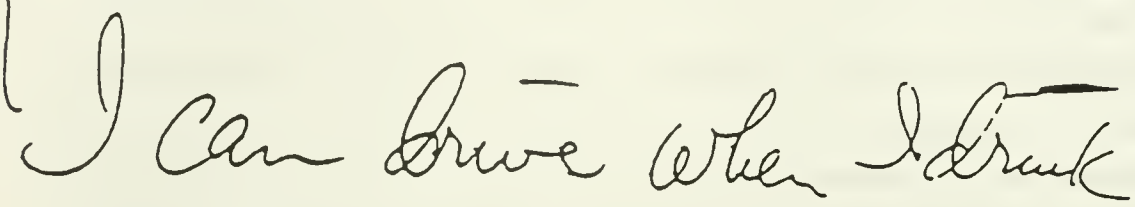
The party begins.

1 

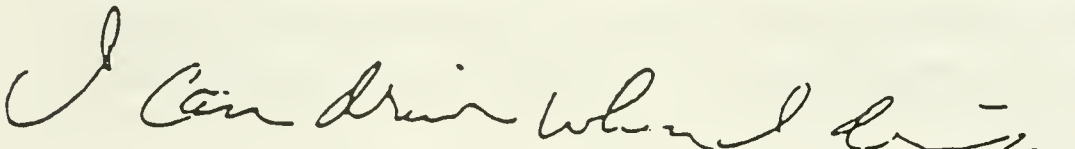
2 drinks later.

2

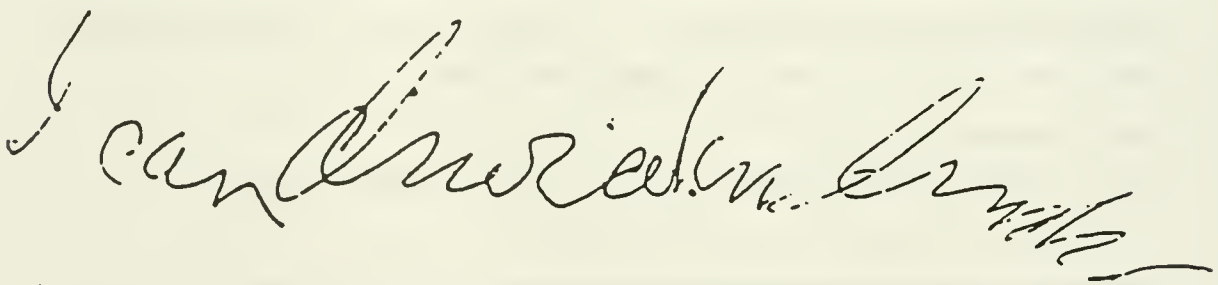
After 4 drinks.



After 5 drinks.

3 

7 drinks in all.

4 

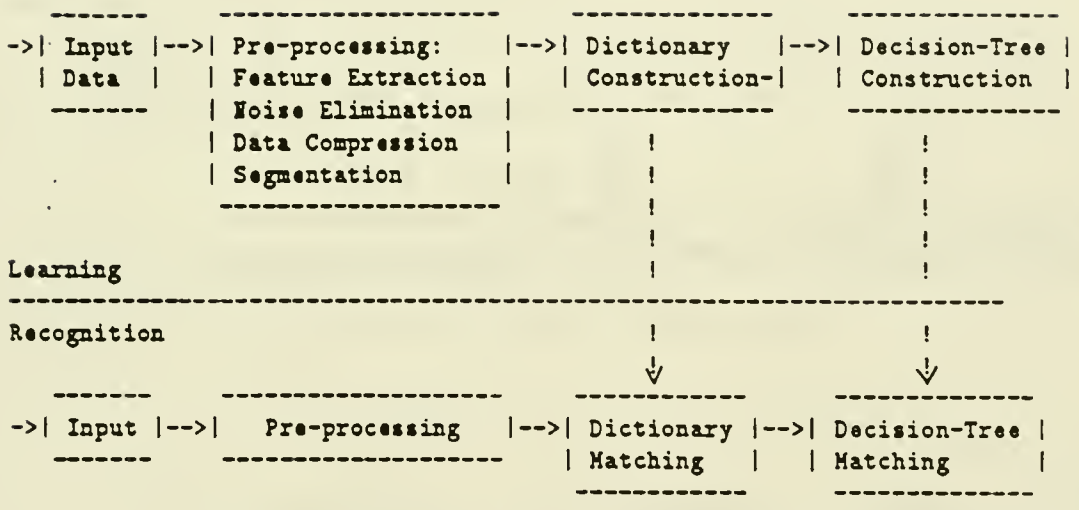


Figure 2.1 Block diagram of a typical recognition scheme.

Figure 2.2 Continuous transformation between X and Y, C and O.

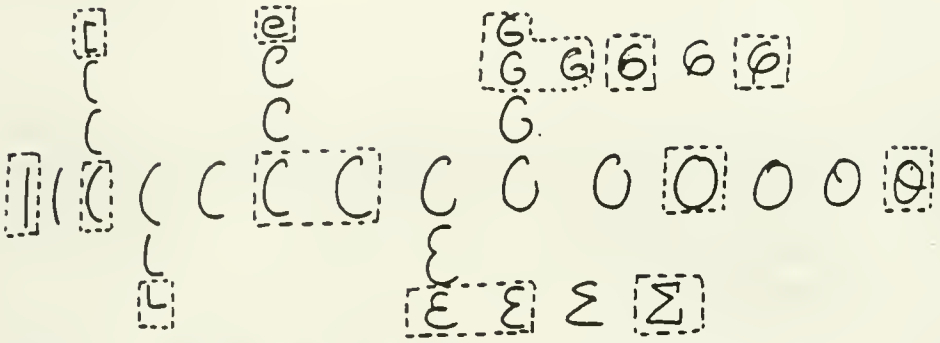


Figure 2.3 Some more examples of continuous transformation

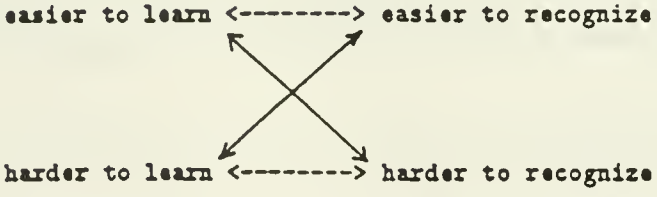


Figure 2.4 Relation between 'learning' and 'recognition'

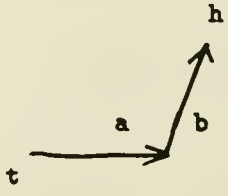
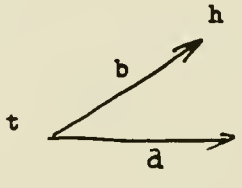
which are easier to learn but harder to recognize and vice versa. Take 'O' and 'Q' for instance. While 'O' is comparatively easier to learn (syntactically) than 'Q', it is harder to recognize (semantically). Notice that Definitions 2.1-2.2 are valid not only for well behaved printed letters, but also for handwriting variations. For instance, in Figure 2.2, DegAmb(X,D,E)$\geq$ 2 and DegAmb(O,D,E)$\geq$ 2.
For more examples, please see Figure 2.3. In general, there are 8 possible relations between learning and recognition as shown in Figure 2.4.

3. Syntactic Methods: PDL, Tree Grammar and Array Grammar

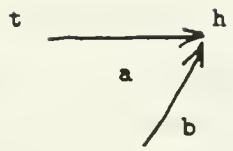
This section investigates several syntactic methods for line-drawing pattern recognition, namely, picture descriptive language, tree grammar and array grammar, with an analysis and comparison between them.

3.1. The picture descriptive language PDL.

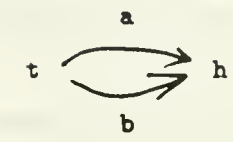
The picture descriptive language (PDL) was developed by Shaw[7,20,21]. Each primitive is labeled at two distinguished points, a tail and a head. A primitive can be linked or concatenated to other primitives only at its tail and/or head.

Primitive Structure Description	Interpretation	Graph
a+b	head(a) CAT tail(b)	
axb	tail(a) CAT tail(b)	

a-b head(a) CAT head(b)



a*b (tail(a) CAT tail(b))
(head(a) CAT head(b))



The grammar that generates sentences in PDL is a context-free grammar

G=(Vn, Vt, P, S)

where Vn={S,Sl}, Vt={b,+,×,-,*,~,/,(,),l}

b may be any primitive (including the "null point primitive" c, which has identical tail and head) and P:

S ⟶b,	S⟶STS,	S⟶~S,	S⟶SL,
S⟶/SL,	SL⟶S,	SL⟶SLTSL,	SL⟶ ~SL,
SL⟶/SL,	T⟶ +,	T⟶ ×,	T⟶ -,
T⟶ *.			

The primitives can be as follows:

h1 → h2 ⟶ g1 ↘ g2 ↘ g3 ↘

d1 ↗ d2 ↗ d3 ↗ v1 ↑ v2 ↑ v3 ↑

Then, we can express the English characters as follows[6].

A→ (d2+((d2+g2)*h2)+g2)

B→ ((v2+((v2+h2+g1+(∼(d1+v1)))*h2)+g1+(∼(d1+v1)))*h2)

C→ (((∼g1)+v2+d1+h1+g1+(∼v1))×(h1+((d1+v1) ×λ)))

D→ (h2*(v3+h2+g1+(∼(d1+v2))))

E→ ((v2+((v2+h2)×h1))×h2)

F→ ((v2+((v2+h2)×h1))×λ)

G→ (((∼g1)+v2+d1+h1+g1+(∼v1))×(h1+((d1+v1+v1-h1)×λ)))

H→ (v2+(v2×(h2+(v2×(∼v2)))))

I→ (v3×λ)

J→ ((((∼g1)+v1)×h1)+((d1+v3)×λ)

K→ (v2+(v2×d2×g2))

L→ (v3×h2)

M→ (v3+g3+d3+(∼v3))

N→ (v3+g3+(v3×λ))

O→ (h1*((∼g1)+v2+d1+h1+g1+(∼(d1+v2))))

P→ ((v2+((v2+h2+g1+(∼(d1+v1)))*h2))×λ)

Q→ (h1*((∼g1)+v2+d1+h1+g1+(∼d1+((∼g1)×g1)+v2))))

R→ (v2+(h2*(v2+h2+g1+(∼(d1+v1)))+g2)

S→((((∼g1)+v1)×h1)+((d1+v1+(∼(g1+h1+g1))+v1+h1+g1+(∼v1))×λ))

T→ ((v3+(h1×(∼h1)))×λ)

U→ ((((∼g1)+v3)×h1)+((d1+v3)×λ))

V→ ((∼g3)×d3×λ)

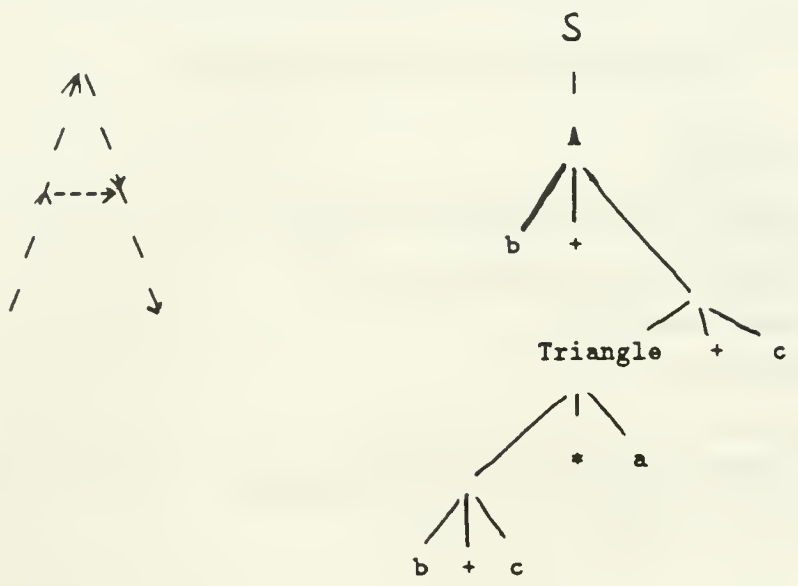
W→ (((∼g3)+d3+g3)+(d3×λ))

X→ (d2+((∼g2)×d2×g2))

Y→ ((v2+((∼g2)×d2))×λ)

Z→ ((d3-h2)×h2)

For example, the character "A" can be described as follows : verbatim



3.2. Tree Grammars[6,11,20]

By extension of one-dimensional concatenation to multidimensional concatenation, strings are generalized to trees. Naturally, if a pattern can be conveniently described by a tree, it can be easily generated by a tree grammar.

Let N^+ be the set of strictly positive integers. Let U be the universal tree domain (the free semigroup with identity element "0" generated by N^+ and a binary operation "."). The depth of a$\in$ U is denoted d(a) and defined as : d(0)=0, d(a . i)=d(a)+1, i$\in N^+$. a $\leq$ b if and only if there exists x $\in$ U such that a . x=b . a and b are incomparable if and only if a $\not\leq b$,*andb* $\not\leq$ a. Figure 3.1 represents the universal tree domain U. D is a tree domain if and only if D is a finite subset of U satisfying(1) b $\in$ D and $a < b$ implies a $\in$ D, and (2)a . j $\in$ D and $i < j$ in N^+ implies a . i $\in$ D.

A ranked alphabet is a pair ($\sum$, r) where $\sum$ is a finite set of symbols and

$$r : \sum \longrightarrow N = N^+ \cup \{0\}$$

For a $\in \sum$, r(a) is called the rank of a. Let

$$\sum_n = r^{-1}(n)$$

A tree over $\sum$ (i.e., over ($\sum$,r)) is a function

$$\alpha : D \longrightarrow \sum$$

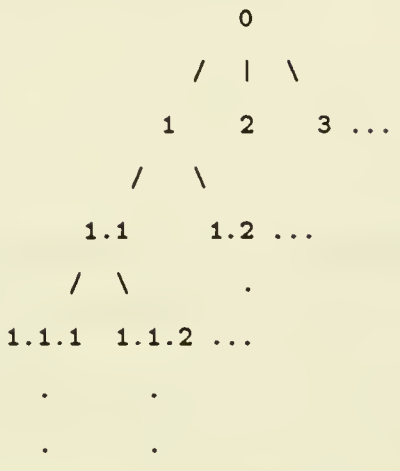


Figure 3.1 Universal tree domain

such that D is a tree domain and

$$r[\alpha(a)] = max\{i \mid a.i \in D\}$$

that is, the rank of a label at "a" must be equal to the number of branches in the tree domain at a. The domain of a tree α is denoted by D(α) or D_α. Let $T_{\sum}$ be the set of all trees over $\sum$. The depth of α is defined as d(α)=max{d(a)|a $\in$ D(α)}. Let α be a tree and "a" be a member of D(α). α/a, a subtree of α at "a", is defined as

$$\alpha/a = \{(b, x) \mid (a.b, x) \in \alpha\}$$

For example, please refer to Fig. 3.2.

$$G_A = (V_A, r_A, P_A, A)$$

where

$$V_A = \{A, A1, A2, N_a, N_d, N_c\}, V_{TA} = \{i, a, c, d\}$$

$$r_A(i) = \{2\}, r_A(a) = \{0, 2\}, \quad r_A(c) = \{0, 1\}, \quad r_A(d) = \{0\}$$

P_A:

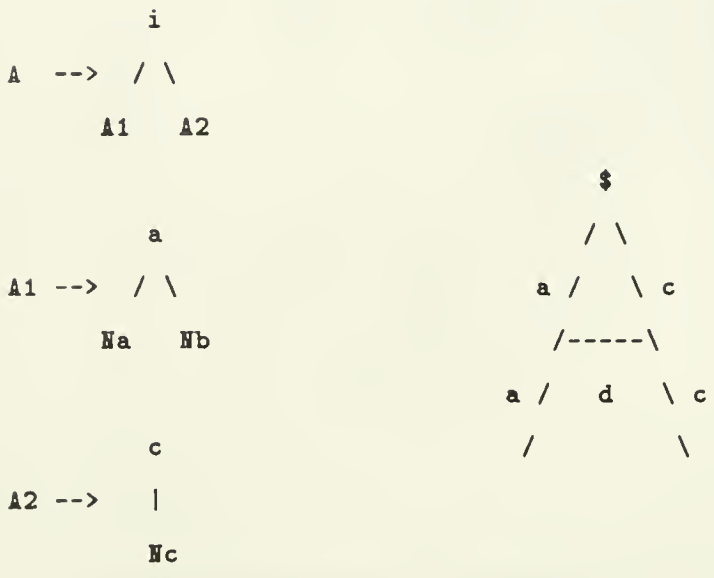


Figure 3.2 A tree grammar for character "A"

3.3. Array Grammar(AG)[4,20,22,33]

Definition 3.1

An isometric array grammar is a quintuple

$$G = (V_N, V_T, P, S, \#)$$

where V_N is a finite nonempty set of nonterminal symbols, V_T is a finite nonempty set of terminal symbols, $V_T \cap V_T = 0$ and $\# \notin (V_N \cup V_T)$ is the background or blank symbol. P is a finite nonempty set of rewriting rules of the form $\alpha \longrightarrow \beta$, where array α and array β

are geometrically identical over $V_N \cup V_T \cup \{\#\}$: α not all #'s. S$\in$N is the starting symbol.

We say that array x directly generates array y, denoted x$\Longrightarrow$ y, if there is a rewriting rule $\alpha \longrightarrow \beta$, x contains α as a subarray and y is identical to x except that the subarray α is replaced with the corresponding symbols of the array β. Let $\Longrightarrow^*$ be the reflexive transitive closure of $\Longrightarrow$. The language generated by an array grammar G, denoted L(G), is the set of all arrays of terminal symbols and #'s that can be generated from the starting symbol S in a field of #'s.

Definition 3.2

The distance between two arrays X and Y over V_T, denoted as d(X, Y) can be defined as the smallest number of error transformations required to derive Y from X.

Example 3.1: Given two arrays over a, b

```
                                                  b
                a a a a a                    a   a a a
                    a                            a
        X =         a        and       Y =       a
                    a                            a
                    a                            b
```

we have

```
                        b                    b                  b
    a a a a a       a a a a a           a   a a a          a   a a a
        a      Ti       a         Td          a      Ts          a
X =     a      |-       a         |-          a      |-          a      = Y
        a               a                     a                  a
        a               a                     a                  b
```

where the symbol |- means "becomes".

The minimum number of error transformations required to transform X to Y is 3. Therefore, d(X,Y) = 3.

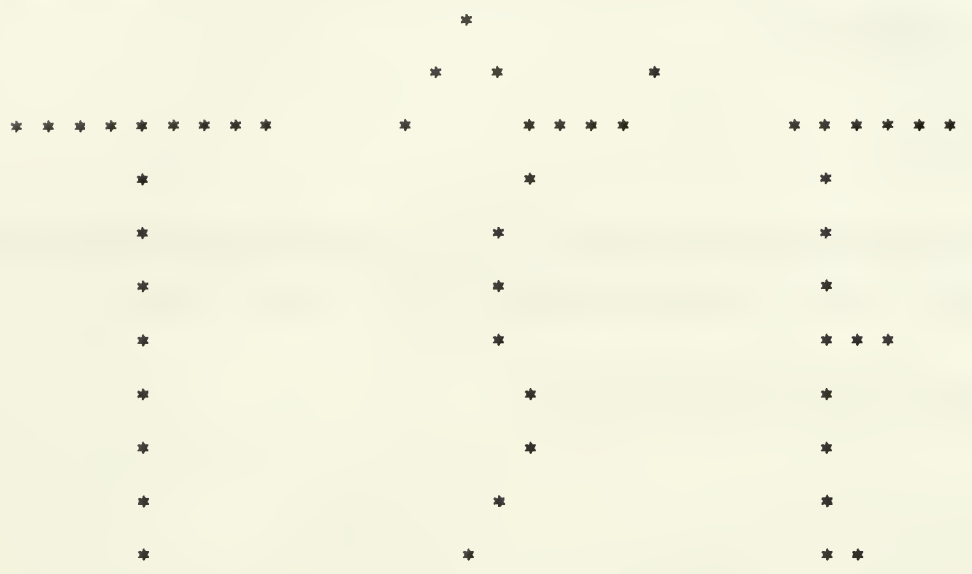


Figure 3.3 Three arrays over ''*"

Definition 3.2 is not very satisfactory as can be seen from the following example.

Example 3.2 Given the 3 arrays in Fig. 3.2, it can be seen that d[(a), (b)] = 18 and d[(a), (c)] = 6. Therefore one would classify (a) and (c), instead of (a) and (b), into one cluster. Anyone who knows English will easily recognize that (a) is more similar to (b) rather than to (c). In fact, a survey of 18 persons showed that all of them thought that (b) is a T while (c) is probably an F or an E.

The above example shows that the direct distance measure between two arrays is not satisfactory for clustering analysis. This motivates trial of the following method.

Assume that every array $\Re$ is a variation or a noisy counterpart of a pure, noiseless array $\Im$, generated by an array grammar G^c, called 'core grammar'. If additional rules are augmented to G^c, resulting in G^a, called 'augmented grammar', then G^a can also generate $\Re$. For instance, in Fig. 3.3, why does a person recognize (b), instead of (c) as a T? Because the person must have learned this to be a T before, and one can assume that a typical, pure or noiseless T was used as a model when he or she learned it. A description of this model for T could be : "T has a horizontal stroke with a vertical stroke attached below the center of the horizontal ones, and both strokes are of similar length". In other words, when a T is learned, it is learned as if there were an array grammar built into one's brain. This grammar can be considered as a core grammar characterizing T. With a certain kind of flexibility, that is with some extra rules attached to this core grammar resulting in a new augmented grammar, one can also recognize some variations or noisy T's, such as, $\widetilde{\int}$, $\widetilde{\{}$ and $\widetilde{\widetilde{\{}}$.

Definition 3.3

The distance between an array $\Re$ and a group of arrays characterized by an augmented array grammar G^a is d_a (G^a ,$\Re$) = minimum number of $\Re$ symbols not covered by the parsing of G^a.

Definition 3.4

The distance between an array $\Re$ and a group of arrays characterized by a core array grammar G^c is d_c (G^c , $\Re$) = minimum number of $\Re$ parsing steps using non-core rules of G^a + d_a (G^a, $\Re$).

Notice that if there does not exist a parsing sequence for $\Re$, then both its d_a and d_c values are either infinite or undefined. In Fig. 3.2, for instance,

$$d_a[D_3,(a)] = 0, d_c[G_2,(a)] = 0, d_a[G_3,(b)] = 0,$$

$$d_c[G_2,(b)] = 9, d_a[G_3,(c)] = -, d_c(G_2,(c)) = -.$$

We now propose an algorithm that will produce an augmented grammar from a core grammar.

Without loss of generality, one can assume that each core grammar is in Chomsky 2-point normal form [4,9], Notice that because a different definition of connectedness is being used, we have a slightly different Chomsky normal form, in that every rule obeys one of the following:

```
                 #     C        #    C        #      C
  A# -> BC,     A  -> B  ,        ->          A ->   B,
                                 A    B

                  A       B       A    B
   #A -> C B,    #    -> C   ,      ->   ,     A   -> B
                                   #    C        #      C
or A ->a
```

where A $\in V_N$, $B, C \in V_N \cup V_T$ and $a \in V_T$.

In general, a rule can be represented by a triple (A, BC,$\overline{d}$), where $0 \leq$ d ≤ 7. For instance, A# $\rightarrow$ BC can be represented by (A, BC,0),

```
                 #        C
                A    --> B
```

can be represented by (A, BC, 1), etc., and A $\rightarrow$ a is represented by (A, a, -). Notice that this representation coincides with the Freeman's chain code octal primitives as to

be shown in Fig. 4.2. of next section.

Algorithm A_l(Core-to-augmented)

Input. k core grammar

$$G_i = (V_{N_i}, V_{T_i}, P_i, S_i, \#), i = 1...k$$

Output. k augmented grammars

$$G_i^a = (V_{N_i}^a, V_{T_i}^a, P_i^a, S_i^a, \#) i = 1...k$$

Step 1. For i=1 to k do step 2

Step 2.

$$V_{N_i}^a = V_{N_i}, V_{T_i}^a = V_{T_i}, S_i^a = S_i$$

and

$$P_i^a = P_i \cup \{(A, BC, (\overline{d}+1)_{mod8}), (A, BC, (\overline{d}+7)_{mod8}) \mid (A, BC, \overline{d}) \in P_i\}$$

For convenience, if in P_i the jth rule is (A, BC, $\overline{d}$), then add j.$(\overline{d}+1)_{mod8}$ and j. $(\overline{d}+7)_{mod8}$ into P_i^a.

Step 3.

$$G_i^a = (V_{N_i}^a, V_{T_i}^a, P_i^a, S_i^a, \#)$$

We are now ready to propose a two-pass clustering procedure.

Cluster Pass I

Input. A set of n digitized patterns X=$\{x_1, x_2, ..., x_n\}$, and a set of k core grammars Z = $\{G_1, G_2, ..., G_k\}$, where L(G_i) $\cap$ $L(G_j)$ = 0 for i $\neq$ j, i, j=1,...,k.

Output. The assignment of x_i, i=1,...,n to k clusters characterized by G_i, i= 1, ..., k.

Step 1. Call CORE-TO-AUGMENTED

Step 2. For i = 1 to n do step 3-5

Step 3. For j = 1 to k do step 4-5

Step 4. Compute $d_a(G_j, x_i)$

Step 5. Assign x_i to cluster p with

$$d_a(G_p, x_i) = min_{j=1...k} d_a(G_j, x_i).$$

Notice that if more than one clusters have the same minimum distance with x_i, then x_i should be assigned to all these clusters.

If x_i has been assigned to more than one cluster, then use Cluster Pass II.

Cluster Pass II

For all $x_{i_1}, x_{i_2}, ..., x_{i_m}$ which are assigned to more than one cluster, do the following.

Input. x_{i_j} and its assigned grammars $G_{i_1}, ..., G_{i_h}$ where 1 ≤ j=m ≤ n, 1 ≤ h ≤ k, 1≤ i_j ≤ n, and 1≤ $i_h \leq k$.

Output. Assignment of x_{i_j} to a proper cluster.

Step 1. Compute $d_c(G_{i_h}, x_{i_j})$ for all G_{i_h} to which x_{i_j} has been assigned.

Step 2. Assign x_{i_j} to cluster i_q where $d_c(G_{i_q}, x_{i_j} = \min\ d_c\ (G_{i_h}, x_{i_j})$ for all G_{i_h} to which x_{i_j} has been assigned.

The set of English characters adapted from Fu and Lu[7] is illustrated using the above two-pass clustering procedure. There are 51 characters from nine different classes: P, X, D, U, F, Y, K, H, and V characterized by G0, G1, G2, G3, G4, G5, G6, G7, and G8, respectively. Eight of the nine classes are selected from four groups, each with similar structures; that is D and P, H and K, U and V and X and Y. The class of character F is different from the other eight classes. Each character is an array on a 20*20 grid, as shown in Fig. 3.4. Generated in a top down fashion, each pattern is parsed upwards from the bottom. The results of 9 clusters are satisfactory as shown in [29].

In Fu and Lu[7], the use of a similarity measure using tree grammar for syntactic patterns in terms of grammar transformations is proposed and a nearest neighborhood

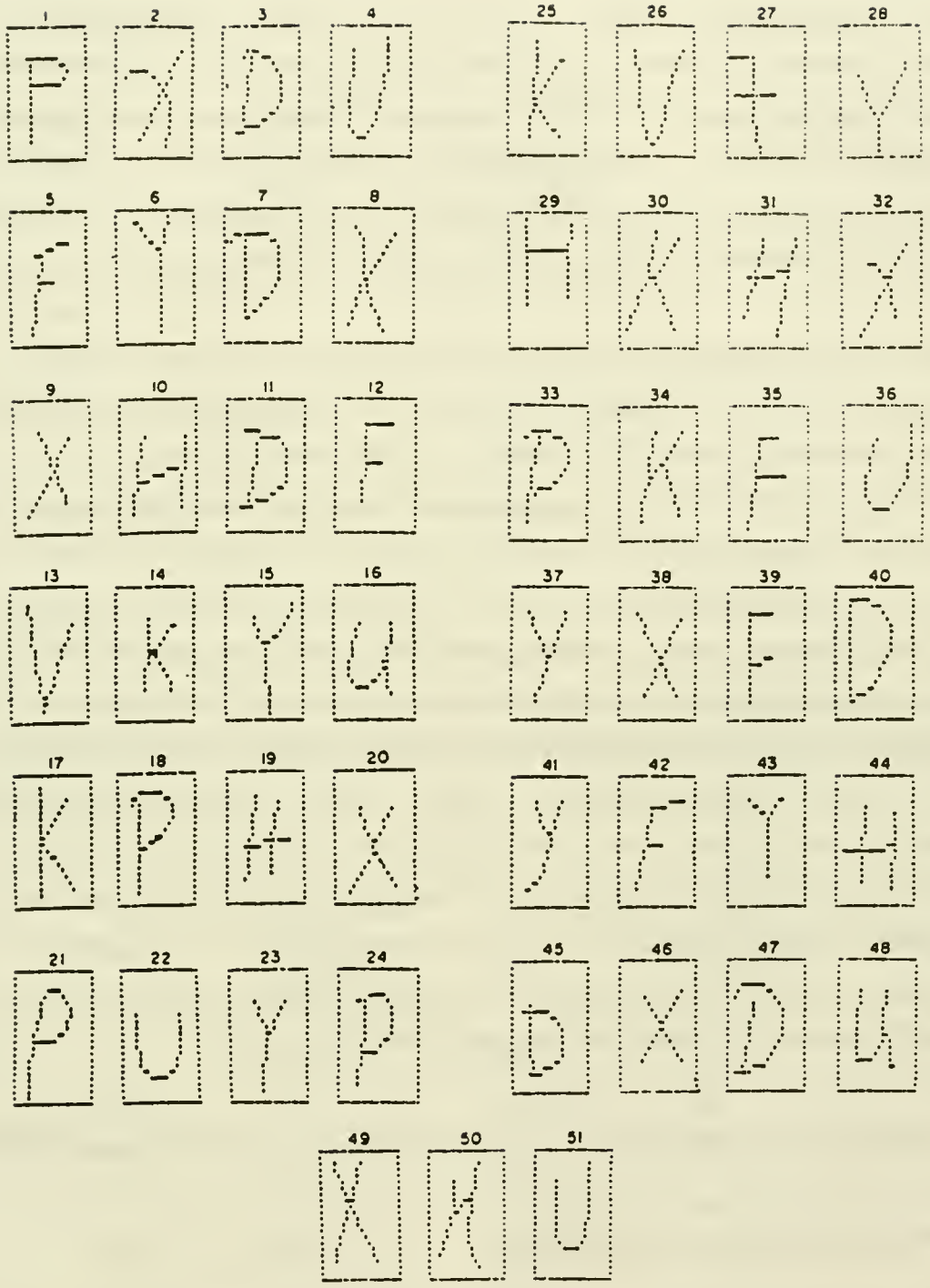


Figure 3.4. 51 digitized English characters.

rule and a clustering procedure are described. This clustering procedure is quite successful for two-dimensional pattern analysis, especially for English characters. However, in this procedure, every character should be first encoded into a one-dimensional string via Freeman's chain code[5] and Shaw's PDL[21]. Further, pattern 47 (Figure 3.4.) is not accurately classified into group 2, D, even when weighted transformations are used with a threshold value as high as 3. In Lu[11], a tree-to-tree distance measurement algorithm is proposed and is applied to clustering analysis for 2-dimensional patterns. Again, in this algorithm, every pattern should be first encoded into a tree representation and, further, there are some confusions between A_5( ) and E_5( ) and between E_4( ) and C_5( ).

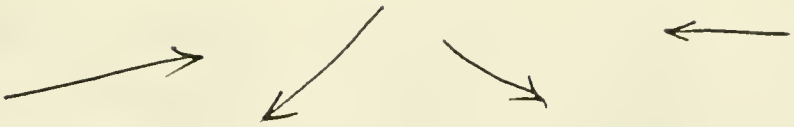
Please note that although the purposes of [29] and [7,11] are somewhat similar, in the sense that they all aim at the same target (i.e. they all try to solve the clustering problem for two-dimensional patterns), [29] employs an entirely different approach. For instance, the model used in[29] is "isometric array grammar"—the first time such a model has been tried for two-dimensional pattern clustering analysis. Also, the definition of distance between two patterns introduced in Section 3 of this paper does not use the conventional error transformation method. Instead, we use the concept of parsing and define the distance between an array and a class of arrays characterized by a Context-Free Array Grammar(CFAG)[4]. Even though the same input data adapted from [7,11] is used to test our two-pass clustering algorithm, we obtain a different result. In addition to the differences discussed above, we also notice that pattern 7 of Fig. 3.4 is classified as an X through our algorithm, in contrast to an F through the algorithm of Fu and Lu[7,11]. A survey of 18 persons showed that 7 of them thought of as an X, 8 of them thought it was an F, while 3 of them thought it was neither. Clearly it depends on how one is trained to write and the array grammar depends on that training.

4. Structural Methods: BM approach and Extented Freeman Approach[EF]

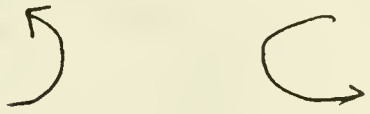
a. Berthod and Maroy's primitives.

We denote Berthod and Maroy's primitives [3] and encoding scheme as BM. Every character is encoded into a string of the 5 primitives shown in Table 4.1.

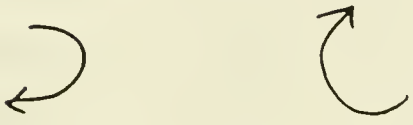
T: straight line



P: positive curve
(counterclockwise)



M: minus curve
(clockwise)



L: penlift

R: cusp

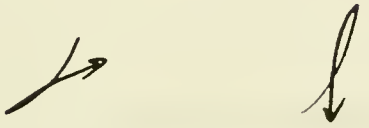


Table 4.1 The five Berthod and Maroy primitives

Examples of codes corresponding to various characters are shown in Figure 4.1.
Let the domain D be all English characters. A dictionary consisting of about 26 characters (strings of primitives) resulting from pre-processing of characters drawn on a graphic tablet is compiled.

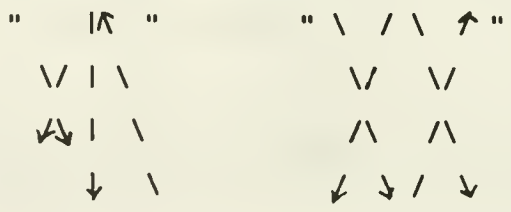
Code	Symbols	Code	Symbols
M	O	TM	S
MLT	J	TLTTT	M
MM	M	TRM	DP
P	C L U	TRLMLT	AF
PLT	EGQ	TRMLTLT	E
PLTLT	E	TRMM	B
PM	S	TRMRM	BR
PRT	G	TRBT	R

PT	G	TRTTLT	AFK
TLM	DPJ	TRTTLTLT	E
TLMM	BR	TRTTT	N
TLMRM	B	TRTTT	M
TLMT	R	TT	LV
TLT	X Y T	TTLT	J
TLTLT	AFHIKNYZ	TTLTLT	E
TLTLTLT	E	TTT	ZNS
TLTT	N	TTTT	WM
TLTTM	B		

Table 4.2 A dictionary of 26 English characters using BM

Now, if the domain D is reduced to capital English characters only, the dictionary will contain 13 ambiguous words (Table 4.2). The most ambiguous word is TLTLT, which has 8 different interpretations. Therefore DegAmb(B,D,BM)=8, B$\in${A,F,H,I,K,N,Y,Z}. The code P has 3 interpretations (C,L,U), TLT has 3 interpretations, and TTT has 3 interpretations (ZNS), etc.

It is seen that characters A,F,H,I,K,N,Y and Z, which are not even syntactically similar, all result in the same encoded word "TLTLT". That makes this word extremely ambiguous and non-informative. A tremendous amount of effort must be spent to disambiguate it. Besides, obviously unacceptable symbols such as



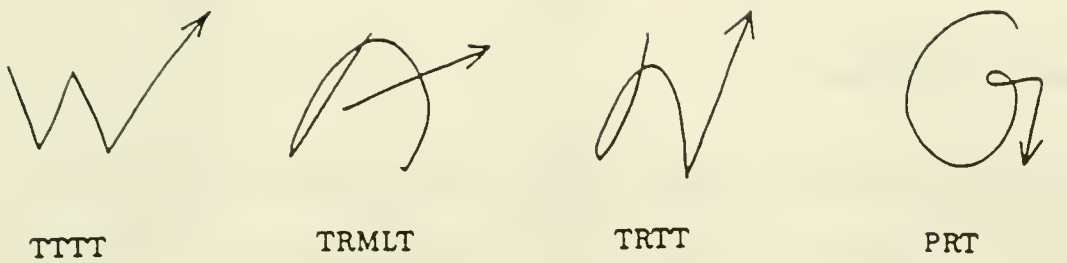


Figure 4.1 Examples of codes corresponding to characters.

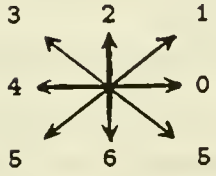


Figure 4.2 Freeman's octal chain code primitives

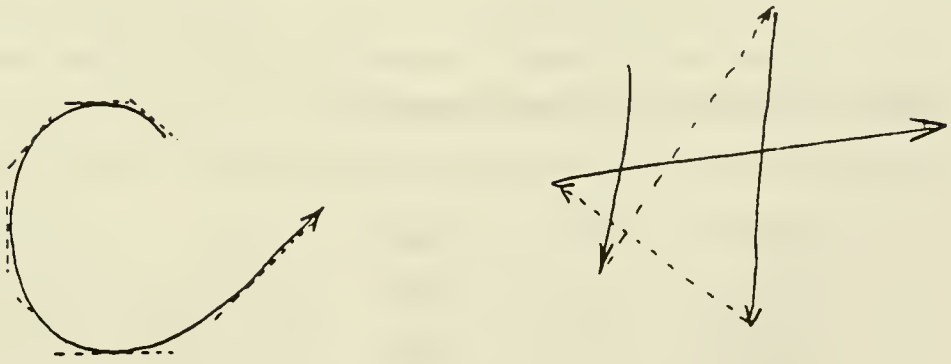


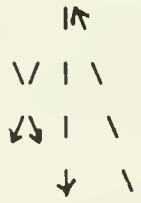
Figure 4.3. (a) Coded word of "C" is 34567011 and (b) Coded word of "H" is 6 $\overline{1}$6 $\overline{3}$0.

could be recognized as an "E", since both have "TTLTLT", which is in the dictionary and is unambiguous.

b. Extented Primitives

We adapt Freeman's chain code[5] and the 8 direction vectors shown in Figure 4.2 as primitives. Each character is encoded into a string of primitives, but only those local extrema points (points which are tangent to one of the 8 vectors) are extracted. The letter "C" could be encoded as 3456701 as shown in Figure 4.3(a). To handle penlift, we consider the vector between 'pen-up' and its successive 'pen-down' and choose the closest octal primitive with a bar on it. Therefore the letter "H" could be encoded as $6\bar{1}6\bar{3}$ 0 as shown in Figure 4.3(b).

Let us call the above described encoding scheme EF(for Extended Freeman) and let D= English characters. Using the same sample data , a dictionary consisting of about 48 words in compiled in an ordering $0 > 1 > \ldots > \bar{6} > \bar{7}$ as shown in Table 4.3. In this dictionary, only 7 words are ambiguous and DegAmb(B, Dm, EF)=2, B$\in${C,O,D,P,X,Y,T}. Further, this method eliminates the possibility of misrecognizing a number of obviously unacceptable symbols. For example,



will not be interpreted as the valid character "E".

$046\bar{2}0\bar{5}0$	E	$076\bar{5}\bar{2}6$	P
$0\bar{4}6$	T	$0\bar{4}6\bar{2}0$	F
$046\bar{2}050$	E	$0\bar{4}64$	J
$0\bar{4}6\bar{4}0$	I	161	N
21	V	2121	W
245670123406	G	2716	M

3456701	CO	4560	C
46$\bar{2}$0	F	46$\bar{2}$0$\bar{5}$0	E
4675	S	560	C
5670123	CO	567012$\bar{3}\bar{6}\bar{7}$	Q
575	S	5$\bar{1}$6$\bar{3}$0	A
5$\bar{1}$7$\bar{1}$5	N	5$\bar{1}$7$\bar{3}$0	A
5$\bar{2}\bar{7}$	XY	61	L
602	U	61$\bar{5}\bar{7}$	K
620657	R	62065765	B
6275	DP	64$\bar{2}$0	J
670	L	67012	U
6$\bar{1}$57	K	6$\bar{1}$6$\bar{3}$0	H
6$\bar{2}$0657	R	6$\bar{2}$065765	B
6$\bar{2}$0765	DP	6$\bar{2}\bar{7}$1	N
6$\bar{2}\bar{7}$16	M	6$\bar{2}\bar{7}$5	DP
6$\bar{2}\bar{7}$575	B	6$\bar{3}$0	T
7171	W	72	V
7$\bar{1}$5	Y	7$\bar{2}$5	X

Table 4.3. An EF Dictionary

5. Improved Extented Freeman Approach[IEF]

In the extended Freeman approach, there are 16 primitives, namely 0, 1, 2, ..., 7, $\bar{0}$, $\bar{1}$, ..., $\bar{7}$. However, we find that this large number of primitives is not required. In this section, the number of primitives is reduced by half by avoiding use of $\bar{0}$, $\bar{1}$, ..., $\bar{7}$. The efficiencies remain the same. In fact, some advantages are gained as shown in Fig. 5.1.

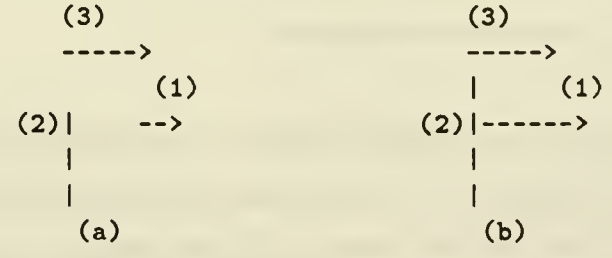


Figure 5.1 (a) EF(0$\bar{4}\bar{6}\bar{2}$0) versus (b) IEF (04620)

0462050	E	076526	P
046	T	04620	F
0462050	E	0464	J
04640	I	161	N
21	V	2121	W
245670123406	G	2761	M
3456701	CO	4560	C
4620	F	0462050	E
4675	S	560	C
5670123	CO	567012367	Q
575	S	51630	A
51715	N	51730	A
527	XY	61	L
602	U	6157	K
620657	R	62065765	B
6275	DP	6420	J
670	L	67012	U
6157	K	61630	H
620657	R	62065765	B
620765	DP	6271	N
62715	M	6275	DP
627575	B	630	T
7171	W	72	V
715	Y	725	X

Table 5.1. An IEF dictionary

6. Off-Line Character Recognition

In this section, off-line character recognition simulating real time on-line technique is discussed and an algorithm for transforming two-dimensional line drawing patterns to parsing sequences based on the "universal array grammar" is constructed.

In conventional syntactic pattern recognition, each class of patterns is represented by a grammar[6]. In order to classify or recognize an input pattern given different classes of patterns characterized by different grammars, the input pattern is normally transformed to a pattern representation through preprocessing. Then it is compared with all given grammars and is classified into the class with the minimum distance measure, provided such a distance is within a certain threshold. Such an approach is basically structural and hierarchical. It divides a rather complicated pattern into subpatterns, which in turn are further divided into subpatterns and so on, until primitives are found which can no longer be divided. It is structurally sound and can overcome some of the difficulties of the decision theoretical (classic) approach discussed earlier.

However, when the number of classes in a pattern recognition problem is very high, pattern matching and classification involve too many grammars and it becomes too time consuming to be practical. Besides, grammatical inference is shown N-P complete hard. Mainly because of this, the concept of "universal grammar" was proposed in [32] for on-line line-drawing patterns, with classification determined by the parsing sequence rather than by grammars.

Such concept of "universal grammar" is explored for off-line line-drawing patterns. We use the model of " array grammar" because it offers several advantages over other methods in dealing with two-dimensional patterns as described in Section 3.

Figure 6.1. patterns 'c' and '2'

Example 6.1. G1=(Vn,Vt,P1,S,#)

Vn={S,S1,S2,S3,S4,S5,S6,
S7,S8,S9,S10}

P1:

0. # S --> S1 *

S1 *

1. # --> S2

S2 *

2. # --> S3

S3 *

3. # --> S4

S4 *

4. # --> S5

S5 *

5. # --> S6

S6 *

6. # --> S7

7. S7 # --> * S8

8. S8 # --> * S9

9. S9 # --> * S10

10. S10 --> *

Example 6.2. G2=(Vn,Vt,P2,S,#)

Vn={S,S1,S2,S3,S4,S5,S6,
S7,S8,S9,S10,S11}

P2:

S1

0. S --> *

1. S1 # --> * S2

2. S2 --> *

S3

3. S3 --> *

S4

4. S4 --> *

S5

5. S5 --> *

S6

6. S6 --> *

S7

7. S7 --> *

S8

8. S8 # --> * S9

9. S9 # --> * S10

10. S10 # --> * S11

11. S11 --> *

It can be seen from Examples 6.1 and 6.2 that arrays generated by G1 and G2 are the character 'C' and digit '2' respectively. Generally speaking, one should construct different array grammars to represent different patterns. It is extremely difficult, if not impossible, to construct adequate grammars for all patterns. Therefore, in [32] a universal line array grammar (ULAG) was proposed to generate all on-line patterns. For recognition, it utilizes the parsing sequence, not the grammar itself, to distinguish between different classes of patterns.

```
ULAG Gu = (Vn, Vt, P, S, #), where Vn = {S}, Vt = {a} and

P:
        0.  S #  -->  a S

              #          S
        1.  S    -->  a

            #          S
        2.  S    -->  a

            #          S
        3.    S  -->    a

        4.  # S  -->  S a

              S          a
        5.  #    -->  S

            S          a
        6.  #    -->  S

            S          a
        7.    #          S

        8.  S    -->  a
```

In this universal line array grammar, a code is associated with each production rule except the terminal rule, and will be produced as part of the parsing sequence if the corresponding rule is successfully applied.

Using ULAG, the parsing sequence of 'C' is 4, 5, 5, 6, 6, 6, 7, 0, 0, 0, 8, and the parsing sequence of '2' is 2, 0, 7, 6, 5, 5, 5, 6, 0, 0, 0, 8.

The above concept can be extended to off-line character recognition.

```
UAG G = (Vn, Vt, P, S, #), where Vn = {S, S1}, Vt = {a} and

  p:
                                                  #            S
      0.  S #  -->  S1 S                  1.   S      -->  S1

          #         S                           #           S
      2.  S    -->  S1                    3.     S    -->     S1

                                                  S            S1
      4.  # S  -->  S  S1                 5.   #      -->  S

          S         S1                          S           S1
      6.  #    -->  S                     7.     #    -->     S

      8.  S1   -->  S                     9.   S      -->  a
```

Example 6.3.

We can use UAG above to get the parsing sequence for English letter E:

```
a a a
a
a a
a
a a a
```

as follows.

```
     4        8        9        4          8 9         6
# S ==> S S1 ==> S S ==> S a ==> S S1 a ====> S a a ==> S1 a a
                                                        S

   8 9          6         8 9 0          9           8
  ====> a a a  ==> a a a  ====> a a a  ====> a a a ====> a a a
        S           S1           a           a           a
                    S            S1S         S1a         S a

   6          86898689          80898089
 ====> a a a  ========>  a a a  =========> a a a
        a                 a                a
        a a               a a              a a
        S1                a                a
                          S1               a a a
```

In the above example, we used UAG parsing sequence to represent a pattern, but these sequences are rather long because some nonterminal to nonterminal parsing and terminal rules were involved. Now, we propose an algorithm, PATSEQ, which produces an unique shorter parsing sequence from a pattern. In order to describe this algorithm, some definitions are required.

Definition 6.1. The neighbors of a pixel, p0, are identified by the eight directors, p1, p2, ..., p8 shown in Figure 6.2.

```
p5 p6 p7
p4 p0 p8
p3 p2 p1
```

Figure 6.2. Pixel p0 and its neighbors

Definition 6.2. The segment of a pixel, p0, is a set of neighbors of p0 which are consecutive.

In the example below, there are two segments, S1 and S2, for the following pixel p0, where S1 =(p5,p6,p7), S2 = (p2,p3).

```
p5 p6 p7
   p0
p3 p2
```

Later, we use Si to indicate the segment i. The length of a segment, length(Si), is equal to the number of elements in the segment Si. For example, length(S1) = 4, length(S2) = 2.

Definition 6.3. A segment is perfect if it satisfies the following conditions:

i) length(Si) $<$ 4, and

ii) pi and pj are not in one segment, where i=2,4,6,8. j=(i+2) mod 8.

Definition 6.4. The center pixel of a perfect segment, Ci, is as follows:

i) If length(Si) = 1 then Ci is the only element in Si.

ii) if 2 $\leq$ length(Si) $\leq$ 3 then Ci=pk, pk is one of the elements of Si (k=2 or 4 or 6 or 8). We also call Ci as the next pixel of p0.

Definition 6.5. The parsing code from the current pixel, p0, to one of its next pixels is the number of the rule successfully applied.

Algorithm PATSEQ: transfer a pattern to a parsing sequence.

Input: a) Initial position (X0,Y0).

b) Digitized pattern in which all segments are perfect.

Output: A parsing sequence.

Method:

Step 1: push(X0); push(Y0); Q:=empty;(Q stores the passed pixels).

Step 2: repeat

Step 3: if stack is empty then step 7.
Y:=pop; X:=pop;
if (X,Y) is in Q then step 3;

Step 4: push all next pixels of (X,Y) and their parsing codes
into stack if no next pixel for P0 then push(-1);

Step 5: Delete current pixel from pattern.

Step 6: z:=pop; if z=-1 then step 6 else print(z).

Step 7: until stack is empty.

The following example illustrates the use of the algorithm PATSEQ.

```
      1  2  3  4  5  6  7
    --------------------------> X
  1 |
  2 |           a
  3 |           b
  4 |           d
  5 |     i  h  e  f  g
  6 v
```

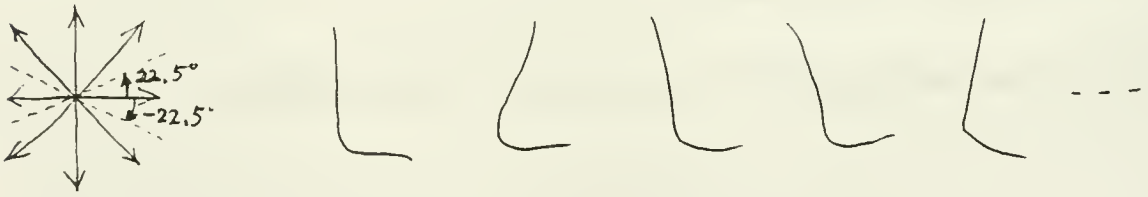
We use character a,b,..., to indicate positions (4,2),(4,3),(5,3),...

The parsing process of character 'E' is as follows.

	Pixel	Stack	Q	Output
1		a		
2	a	b 6	a b	6
3	b	d 6 c 0	a b c d	0
4	c	d 6 -1	a b c d	6
5	d	e 6	a b c d e	6
6	e	h 4 f 0	a b c d e f h	0
7	f	h 4 g 0	a b c d e f g h	0
8	g	h 4 -1	a b c d e f g h	4
9	h	i 4	a b c d e f g h i	4
10	i	-1	a b c d e f g h i	

Please note that in this method, each parsing sequence code represents a line segment vector that can cover -22.5 to +22.5 degrees.

Example 6.4. The letter "L" written in the following ways are all transformed in the same parsing sequence "66600". Therefore, only one learning sample is necessary for this letter, and it takes only one address in the dictionary. This address can be quickly retrieved during the recognition pattern matching process.



This technique significantly lowers the number of learning samples and the size of the dictionary. It also saves pattern matching time because each parsing sequence can actually function as a hashing code that serves as an address, which can represent a rather large class of patterns in the dictionary.

7. Discussions and Conclusions

We introduced the basic concept of degrees of recognizability and ambiguity. Their relationship has been discussed. Although there are many other factors such as human factors, social environment, writing instruments, software and hardware environment as well as algorithm-oriented characteristics such as segmentation, data reduction, resolution, primitive selection, thresholding and quantization, nevertheless, the degree of ambiguity plays an inherent role as an important criterion of recognizability for handwritten symbols.

Two different categories of experiments representing different recognition schemes for on-line handwritten symbols were conducted. The first used basic structural shapes as primitives while the second used octal chain codes as primitives. The second method provided advantages of flexible sizes, orientation, variations, and the need for fewer learning samples. It also possesses an inherently lower degree of ambiguity. Besides, in this method, it is less likely to mis-recognize an obviously unacceptable symbol as a valid one. A summary is depicted in Table 7.1.

	Grammar	Structure	Accuracy (degrees of ambiguity and recognizability)
TG	Simpler	Less Flexible	Low
AG	More Complex	More Flexible	High
PDL	Context Free	Flexible	Medium
BM	---	Flexible (Fewer Primitives)	More Ambiguous
EF	---	Flexible (More Primitives)	Less abmiguous
IEF	---	Flexible (Few Primitives)	Less ambiguous

where TG: Tree Grammar, AG: Array Grammar, PDL: Picture Descriptive Language
BM: Berthod and Maroy, EF: Extended Freeman, IEF: Improved EF

Table 7.1 A summary of comparisons of different methods

References

1. Ahmed, P. and Suen, C.Y., Computer Recognition of Totally Unconstrained Handwritten Zip Codes, *Int. J. of Pattern Recognition and Artificial Intelligence (IJPRAI)*, vol. 1, no. 1, 1-15 (1987).

2. Baird, H., Simon, K., and Pavlidis, T., Components of an Omnifont Reader, *Int. Symp. on Pattern Recognition*, John Wiley and Sons, 343-362 (1986).

3. Berthod,M. and Maroy, J.P., . Learning in Syntactic Recognition of Symbols Drawn on a Graphic Tablet. *Comput. Graphics and Image Processing* 9, 166-182, (1979).

4. Cook, C. R. and Wang, P. S. P., A Chomsky Hierarchy of Isotonic Array Grammars and Languages, *Comput. Graphics Image Process*, vol. 8, no. 4, 144-152 (1978).

5. Freeman, H. On the Encoding of Arbitrary Geometric Configurations, *IRE Trans. Electronic Comput.* 10, 206-268 (1968).

6. Fu, K. S., *Syntactic Pattern Recognition and Applications*, Englewood Cliffs, N.J., Prentice-Hall, (1982).

7. Fu, K.S. and Lu, S.Y., A Clustering Procedure for Syntactic Patterns. *IEEE Trans. Systems Man. Cybernet*, 7, 734-742 (1977).

8. Gupta, A., Hazarika, S., Kallel, M., and Srivastava, P., Optical Image Scanners and Character Recognition Devices: A Survey and New Taxonomy, MIT IFSRC Report No. 107-89, (1989)

9. Hopcroft, J.E. and Ullman, J.D.. *Introduction to Automata Theory, Languages, and Ccomputation.* Addison Wesley, New York, (1979).

10. Kushnir, M. et al., Recognition of Handprinted Hebrew character using Features Selected in the Hough Transform Space, *Pattern Recognition*, vol. 18. no. 2, 103-114, (1985).

11. Lu, S. Y., A Tree-to-Tree Distance and Its Application to Cluster Analysis, *IEEE Trans. Pattern Anal. Mach. Intell.* PAMI-1, vol. 1, no. 2, 219-224 (1979).

12. Mantas, J., An Overview of Character Recognition Methodologies, *Pattern Recognition*, vol. 19, no. 6, 122-134 (1986).

13. Masuda, I. et al, Approach to Smart Document Reader System, *IEEE-CUPR Proc.* 550-557, (1985).

14. Mori, S. et al., Research on Machine Recognition of Handprinted Characters, *IEEE-PAMI.* vol.6, no. 4, 386-405 (1984).

15. Morishita, T., Ooura, M. and Ishii, Y., A Kanji Recognition Method which Detects Writing Errors, *Int. J. of Pattern Recognition and ArtificialIntelligence(IJPRAI)*, vol. 2, no. 1, 181-195 (1988).

16. Nagahashi, H. Nakatsugama,M., A Pattern Description and Generation Method of Structural Character. *IEEE-PAMI.* vol. 8, no. 1, 112-117 (1986).

17. Nandhakumar, N., and Aggarwal, J., The Artificial Intelligence Approach to Pattern Recognition - A Perspective and Overview, *Pattern Recognition*, vol. 18, no. 6, 383-389 (1985).

18. Pavlidis, T., *Structural Pattern Recognition*, Springer-Verlag,New York, (1980).

19. Rich, E., *Artificial Intelligence*, McGraw-Hill, (1983)

20. Rosenfeld, A, *Picture Languages: Formal Models for Picture Recognition*, Academic Press, New York (1979).

21. Shaw, A.C., A Formal Picture Description Scheme as A Basis For Picture Processing Systems, *Inf. Control* vol. 14, 9-52 (1969).

22. Siromoney, R., Array Languages and Lindenmayer Systems - A Survey, *The Book of L*, Rozenberg, G. and Salomaa, A. (Ed.), Springer Verlag (1985).

23. Suen, C.Y., Factors Affecting the Recognition of Handprinted Characters. *Proc. Int. Conf. Cybernetics and Society*, 174-175 (1973).

24. Suen, C.Y., Berthod, M. and Mori, S., Automatic Recognition of Handprinted Characters - The-state-of-the-art, *Proc. IEEE* vol. 68, no. 4, 469-487 (1980).

25. Stern, R., From Intelligent Character Recognition to Intelligent Document Processing, *Proc. of Int. Electronic Imaging Exposition and Conference*, Prentice-Hall, 236-245 (1987).

26. Tanimoto, S., *The Elements of Artificial Intelligence*, Computer Science Press, (1987).

27. Tappert, C.C., Suen, C.Y. and Wakahara, T., On-Line Handwriting Recognition-A Survey, *Proc. ICPR'88*, Rome, 1123-1132 (1988).

28. Ullmann, J., Advances in Character Recognitions, in *Applications of Pattern Reconition.* Ed. by K. S. Fu, CRC 197-236 (1982).

29. Wang, P.S.P., An Application of Array Grammars to Clustering Analysis. *Pattern Recognition*, vol. 17, 403-410 (1984).

30. Wang, P.S.P., *Dual Level Pattern Recognition System - A New OCR Technique*, US Patent PN 4521909 (1985).

31. Wang, P.S.P., A New Character Recognition Scheme With Lower Ambiguity and Higher Recognizability. *Pattern Recognition Letters* 3 431-436 (1985).

32. Wang, P.S.P., On-line Chinese Character Recognition, *International Conference on Electronic Imaging* 209-214. (1988).

33. Wang, P.S.P.(Ed.) *Array Patterns, Grammars and Recognizers* World Scientific Publishing Co., (1989).

34. Wang, P.S.P., A More Natural Approach for Recognition of Line-Drawing Patterns, *Image pattern recognition: algorithm implementations, techniques and technology*, *Proc. SPIE*, vol. 755, 141-160, (1987).

35. Winston, P. H., Artificial Intelligence, Addison-Wesley, (1984).

36. Winston, P.H.(ed) with Shellard S., *Artificial Intelligence at MIT - Expanding Frontiers*, MIT Press, to appear (1990).

Date Due

Lib-26-67

MIT LIBRARIES

3 9080 00698826 2